LA TOMA DE DECISIONES

No todos los problemas se resuelven linealmente, a veces es necesario tener que tomar una decisión o ejecutar determinadas acciones cuando una condición se cumple, y otras cuando no lo hace. Supongamos que nuestro problema consiste en mantener la temperatura de un balde con agua tibia. Para hacerlo nosotros podemos agregar agua caliente o agua fría. En este problema necesitaríamos tomar una decisión sobre qué tipo de agua agregar. De igual forma, hay muchos problemas en los que para poder resolverlos necesitamos conocer una condición o tomar una decisión. En C# es sencillo poder lograr esto, ya que el lenguaje nos provee diferentes herramientas para poder lograrlo. Tendremos que utilizar expresiones y éstas se evaluarán. En este caso usaremos expresiones relacionales y expresiones lógicas, que se evaluarán, y dependiendo del resultado de esa evaluación, se llevarán a cabo ciertos pasos del algoritmo u otros. Empecemos por conocer las expresiones que necesitamos.

Expresiones relacionales

Las expresiones relacionales se usan para expresar la relación que existe entre dos valores. Los valores pueden estar contenidos adentro de variables o ser colocados explícitamente. Estas expresiones, al igual que las expresiones aritméticas, tienen sus propios operadores. La expresión será evaluada, pero el resultado de la evaluación tendrá únicamente dos valores posibles: true o false.

Ambos valores son de tipo bool y true es usado para indicar que la expresión evaluada es verdadera. El valor de false por su parte se utiliza para indicar que la expresión evaluada es falsa.

Empecemos por conocer primero a los operadores relacionales y luego veremos ejemplos de expresiones relacionales con su resolución.

Operadores relacionales

En la tabla 1 podemos apreciar los operadores relacionales en C#. La forma como están escritos sus signos es la forma como debemos colocarlos en nuestro programa.

Un error muy común que sucede cuando se empieza a programar en C# es el de confundir el

operador de igualdad con el de asignación. Si se obtiene un error en una expresión, hay que verificar esto. No olvidemos que la asignación lleva un signo igual y la igualdad doble signo igual.

SIGNO	OPERADOR
	Igualdad
	No igual
	Mayor que
	Menor que
	Mayor que igual
	Menor que igual

Tabla 1. Esta tabla nos muestra los diferentes operadores relacionales de C#.

El operador de igualdad se representa con dos signos igual juntos y sin espacio. Colocar un solo signo igual, indica que el operador será de asignación. Esto puede derivar en errores de lógica, porque una asignación siempre evalúa a verdadero, pero una igualdad no. Este operador es el más sencillo. Supongamos que ya tenemos las variables y las inicializamos con los siguientes valores:

```
int
int
int
int
```

Ahora crearemos nuestra primera expresión y la evaluaremos:

```
a == c
```

Como el valor contenido en a es 5 y el valor contenido en c también es 5, vemos que se cumple 5 igual a 5, por lo que la expresión se evalúa como true. Veamos qué sucede en la siguiente expresión:

```
osa
```

En este caso el valor contenido en d es 4, por lo que la expresión 5 igual a 4 no se cumple y el valor de la expresión es false. En el lado derecho y en el lado izquierdo de la expresión podemos colocar cualquier valor, variable o expresión. Si colocásemos una expresión, primero se evaluaría ésta y luego se procedería a evaluar la expresión relacional principal.

```
a == (3 + 2)
```

En este caso 3 + 2 se evalúa como 5 y luego 5 igual a 5, lo que nos da como valor final de la expresión true. Otro ejemplo es el siguiente:

```
(b – 2) == d
```

Al evaluar b - 2 obtenemos 5, entonces la expresión es 5 igual a 4, lo que evidentemente nos da como resultado final false.
Esta es la forma como se evalúan las expresiones relacionales y ahora veremos más ejemplos con otros operadores. También tenemos al operador de desigualdad, que se crea por medio de un signo de admiración y luego el signo de igual.

```
a != 7
```

En este caso tenemos 5 no es igual a 7, lo cual es cierto, y obtenemos el valor true como resultado de la evaluación.

```
a != c
```

Para esta expresión tenemos 5 no es igual a 5, y se evalúa la expresión como falsa, lo que nos da false como valor final. El operador > sirve para evaluar una relación del tipo mayor que. Si el valor del lado izquierdo es mayor que el valor del lado derecho, regresará true, en caso contrario regresará false.

```
b > a
```

Esto es 7 mayor que 5 y da como resultado true.

```
a > b
```

En este caso 5 mayor que 7 es falso y el resultado de la expresión es false.

```
b > b
```

Este expresión nos da 6 mayor que 6, lo cual, si analizamos todos los pasos hasta aquí, nos indica que es falso también y obtenemos false como resultado.

En el operador < evaluamos una relación del tipo menor que. Esta da true como resultado si el valor del lado izquierdo es menor que el valor que encontramos en el lado derecho. Si esto no se cumple, entonces el valor regresado será false.

aza

Tenemos 4 menor que 5, lo que resulta cierto y el resultado es true.

3)

La expresión a evaluar es 4 menor que 2, y se tiene false como resultado de la expresión.

aza

Esta expresión también da como resultado false, ya que 4 menor que 4 no es verdadera. Para poder evaluar si un valor es mayor que o igual a otro valor usamos el operador Su forma de trabajo es similar a la de los otros operadores.

```
b >= a
```

Para esta expresión tenemos 7 mayor que igual a 5, es cierta y da true de resultado.

```
b >= (a + c)
```

En este caso obtendríamos 7 mayor que igual a 10, lo cual sabemos que es falso y y, por lo cual, como resultado obtendríamos false.

Los operadores que se escriben con dos signos deben escribirse correctamente. No es lo mismo

>=que Esto puede llevarnos a errores de sintaxis, pero al escribirlos en la forma correcta se verán corregidos. También debemos recordar que se escriben sin espacios entre los signos, ya que esto nos lleva a otro error de sintaxis.

```
b >= b
```

Aquí comparamos la diferencia de resultado con relación a usar >= en lugar de >. Para esta expresión evaluamos 7 mayor que igual a 7. Esto es verdadero y el resultado final es true. De igual manera tenemos al operador <=, pero éste evalúa a la inversa.

```
a <= b
```

La expresión a evaluar es 5 menor que igual a 7, y obtenemos true como resultado.

```
(a + c) <= b
```

En este caso tenemos 10 menor que igual a 7 y resulta falso.

```
(a + 2)<= b
```

En esta última expresión 7 menor que igual a 7 es verdadero y nuevamente obtenemos true como resultado.

El uso de if

Ya aprendimos a usar las expresiones relacionales y el tipo de valor devuelto. Es el momento de hacer algo práctico con ellas. Este tipo de expresiones se usa en diferentes casos, pero generalmente en las estructuras selectivas o repetitivas.
Las estructuras selectivas serán estudiadas en este capítulo, las repetitivas en un capítulo posterior. Las estructuras selectivas son aquellas que nos permiten hacer una selección entre dos o varias rutas de ejecución posibles. La selección se llevará a cabo según el valor de una expresión. Esta expresión puede ser una expresión relacional. Nuestra primera estructura selectiva se conoce como if, que es un si condicional. Si tal cosa sucede, entonces haz tal cosa. El uso del if es sencillo:

```
if (expresión)
    Sentencia a ejecutar
```

El uso de if requiere que coloquemos una expresión a evaluar entre paréntesis. Si el resultado de esta expresión es true, entonces la sentencia a ejecutar se lleva a cabo.
Si el resultado de la evaluación es false, entonces la sentencia a ejecutar nunca se lleva a cabo, o dicho de otra forma, es ignorada.
En el diagrama de flujo if se representa por medio de un rombo. En el interior del rombo colocamos la expresión a evaluar, de las esquinas sacamos una ruta de

ejecución en el caso de que sí se cumpla la condición o en el caso de que no se cumpla.

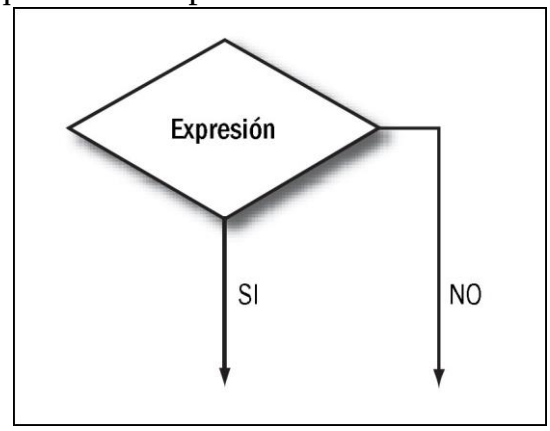

Figura 1. Este rombo simboliza a if con la expresión en su interior.

Veamos un primer ejemplo donde nos puede servir if y las expresiones. Crearemos un programa que le pida al usuario un número, y la computadora debe decir si el número es positivo o negativo. Para ver el algoritmo de este programa, creamos su diagrama de flujo e incluimos los if necesarios.

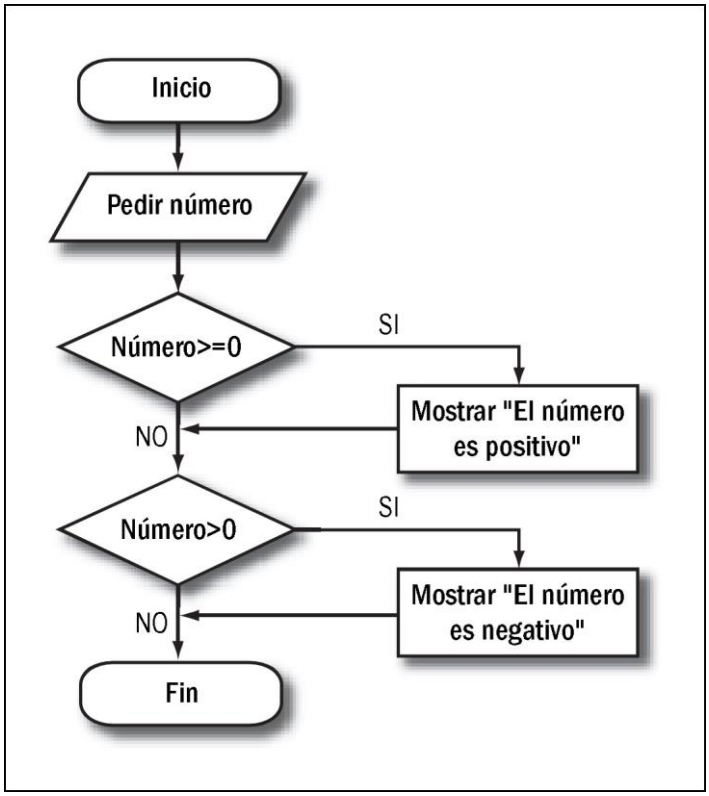

Figura 2. Éste es el diagrama de flujo del algoritmo para resolver el problema. Podemos observar que tenemos dos rutas de ejecución en cada if.

```
using System; using
System.C011ections.Generic;
using System.Text;

namespace AplicacionBase

class Program

// Esta es la función principal del programa
// Aquí inicia la aplicación static void
Main(string[] args)

int numero -- 0; // Donde guardamos el número string valor = // Para
guardar la cadena dada por el usuario

// Pedimos el número
Console.Write("Dame un numero entero:" ) ;
Valor = Console.ReadLine() ; numero = Convert.ToInt32(va10r) ; //
Convertimos la cadena a entero

// Hacemos uso de if con la expresión para el caso de los positivos if
(numero       0)
Console WriteLine("El numero {0} es positivo", numero) ; // se ejecuta Si se
    cumple numero>=0

// Hacemos uso de if con la expresión para el caso de los negativos if
(numero < 0)
Console WriteLine("El numero {0} es negativo", numero) ; // se ejecuta Si
    se cumple numero<0
```

Cuando hacemos uso de if, nunca debemos colocar punto y coma después del paréntesis ya que

hacerlo no es un error de sintaxis sino de lógica. El compilador interpreta esto como si deseamos que se ejecute una sentencia vacía cuando if se cumple. Si vemos que la sentencia if siempre se cumple sin importar cómo se evalúa la expresión, es posible que tengamos este error.

```
        }
    }
```

La primera parte el programa es de muy fácil comprensión. Luego de ésta, nos encontramos con la primera sentencia if. Para este if se tiene la expresión numero 0. Si esa expresión se evalúa como true, entonces la siguiente sentencia se ejecuta, de lo contrario será ignorada y continuará el programa.
Supongamos que el valor guardado en numero es 3, entonces la expresión relacional regresa un valor de true. Al obtener true, if lleva a la ejecución y el mensaje que dice que se ejecuta un número positivo.
Luego se continúa con el siguiente if. Para éste, la expresión es numero < 0, como en este ejemplo numero vale 3, entonces la expresión se evalúa como false, por lo que, mientras el valor de la expresión sea false, if no ejecutará en ningún momento el mensaje que dice que el número es negativo. Esto es exactamente lo que deseamos que realice desde un principio este programa.

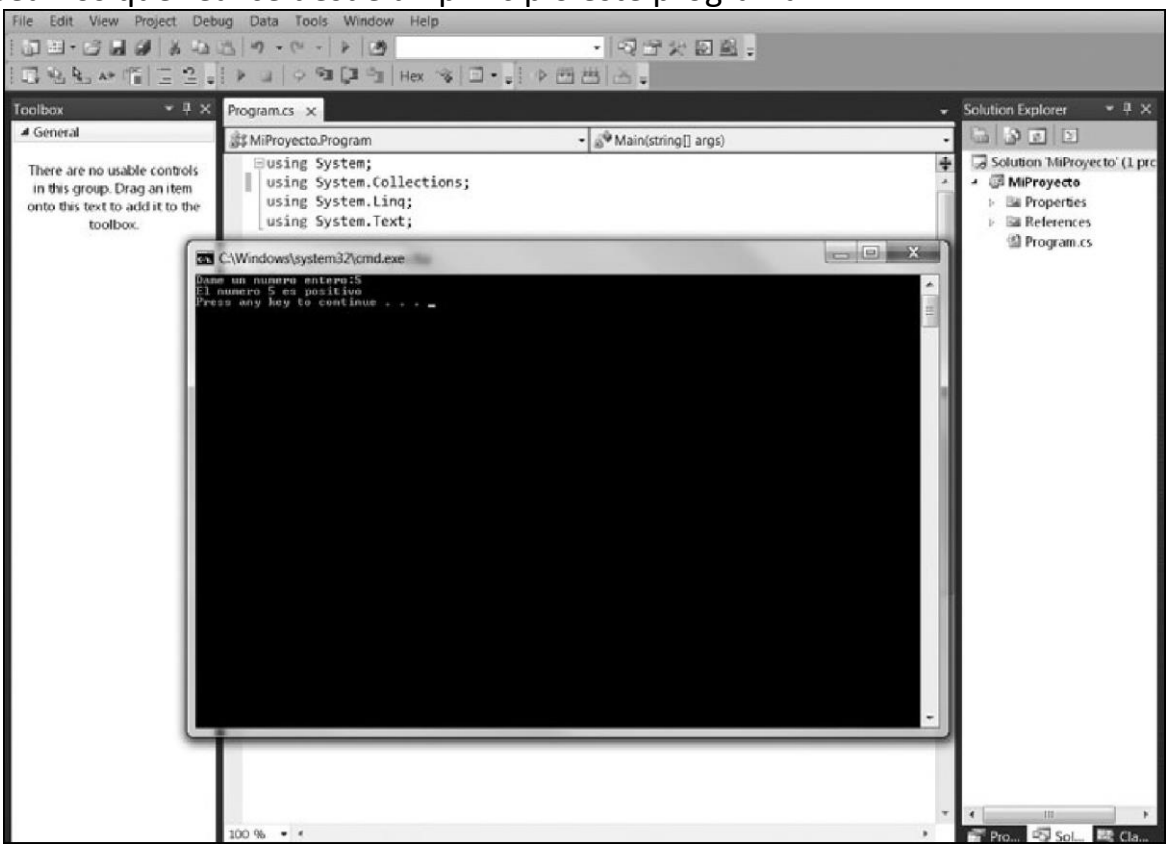

Figura 3. Éste es el resultado de la ejecución
cuando pasamos un número positivo al programa.

Este es un buen momento para comenzar a experimentar en nuestros desarrollos, cambiando los valores, tanto negativos como positivos, incluido el cero y ver cómo responderá nuestro programa.

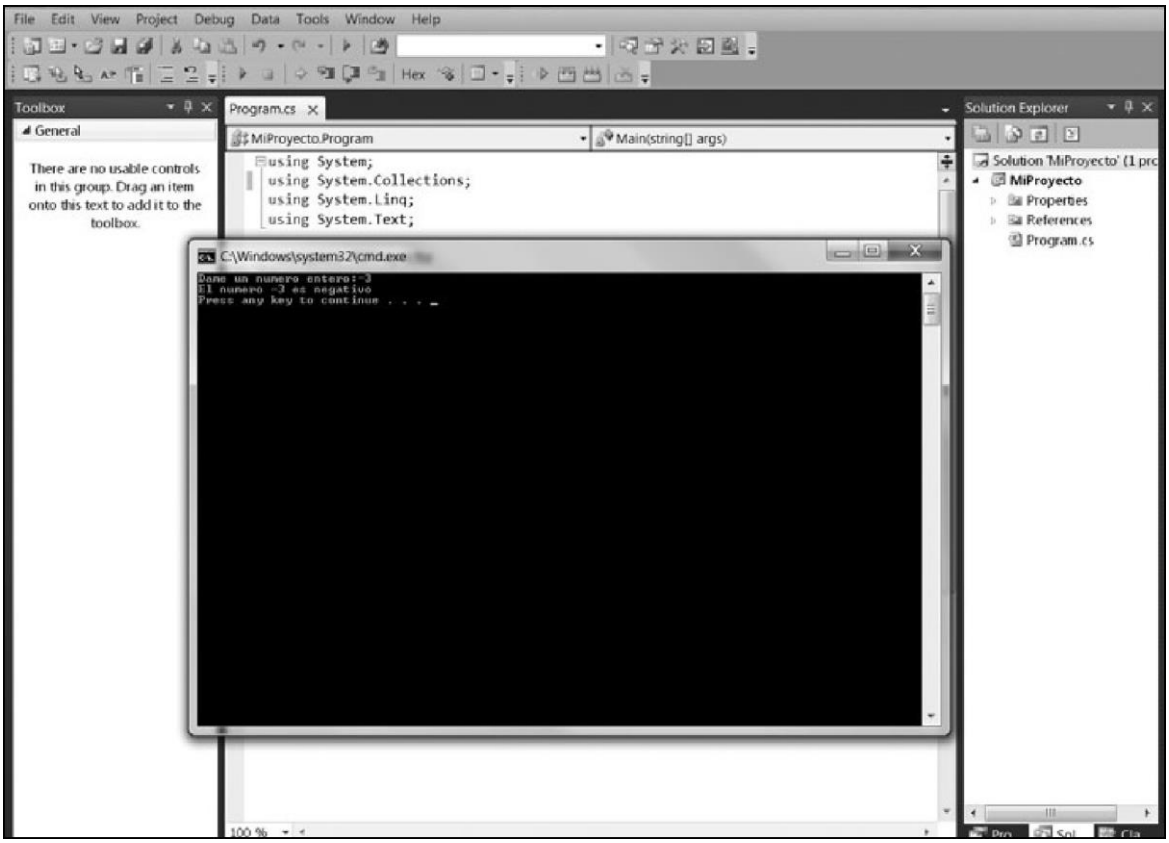

Figura 4. En este caso damos un número negativo y podemos observar el comportamiento del programa.

Bloque de código con if

Como vimos, if efectivamente puede sernos de mucha utilidad para que el programa lleve a cabo la decisión correcta, sin embargo, solamente ha podido ejecutar una sentencia. Con frecuencia nos encontraremos con el caso cuando necesitamos que se ejecute más de una sentencia si la condición se cumple. Una buena opción para resolver este inconveniente encontrado es colocarle un bloque de código a if. En este caso, el bloque de código se ejecutará siempre que la expresión dentro de if sea evaluada como true, y el bloque de código completo será ignorado siempre que la expresión sea evaluada como false.

El bloque de código es un conjunto de sentencias agrupadas y debe ser abierto con { y cerrado con }, quedará compuesto de la siguiente forma:

```
if (expresión)
    Sentencia
    Sentencia

    Sentencia 2;

              n;
```

Veamos un ejemplo donde hacemos uso de esto. Sabemos que la división entre cero no está definida, entonces debemos hacer un programa que le pregunte al usuario el dividendo y el divisor, pero que cuando el divisor sea cero no lleve a cabo la división. Como siempre, primero hacemos nuestro análisis y creamos el algoritmo en forma de diagrama de flujo.

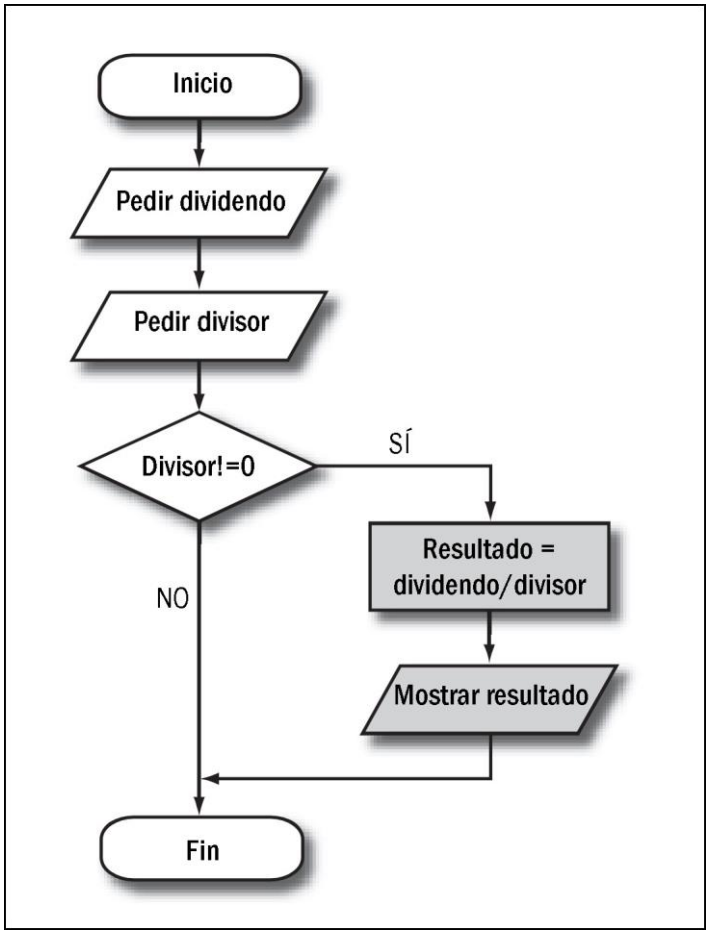

Figura 5. Éste es el diagrama de flujo del algoritmo. Podemos pensar que lo que hemos agrupado es lo que se encuentra adentro del bloque de código de if.

El programa queda de la siguiente forma:

```
using System; using
System.C011ections.Generic;
using System.Text;

namespace AplicacionBase

class Program
    {
     // Esta es la función principal del programa
```

```
// Aquí inicia la aplicación static
void Main(string[] args)

    // Variables necesarias
    float dividendo = 0 Of
    float divisor = 1 Of
    float resultado = 0 Of
    string valor =

    // Pedimos el dividendo
    Console.WriteLine("Dame el dividendo:" )
    ; valor = Console.ReadLine() ; dividendo =
    Convert .ToInt32(va10r) ;

    // Pedimos el divisor
    Console.WriteLine("Dame el divisor:" ) ;
    valor = Console.ReadLine() ; divisor --
    Convert .T01nt32(va10r) ;

    // Si el divisor es válido, entonces hacemos la
    división if (divisor != 0.0f)

        // Hacemos la operación
        resultado = dividendo / divisor;

        // Mostramos el resultado
        Console.WriteLine("E1 resultado de la división es {0}" ,
            resultado) ;
```

Podemos observar que hemos colocado un bloque de código para if. Si la expresión de divisor 0.0f es verdadera, entonces se ejecuta el bloque de código donde tenemos la operación y el desplegado del resultado. Si la expresión se evalúa como falsa, todo el bloque de código es ignorado y no ejecutan las sentencias que están en su interior. Ahora podemos compilar el programa y dar valores para hacer pruebas y observar el comportamiento. Vemos que efectivamente cuando el divisor tiene un valor de cero, la operación y el mensaje no se ejecutan.

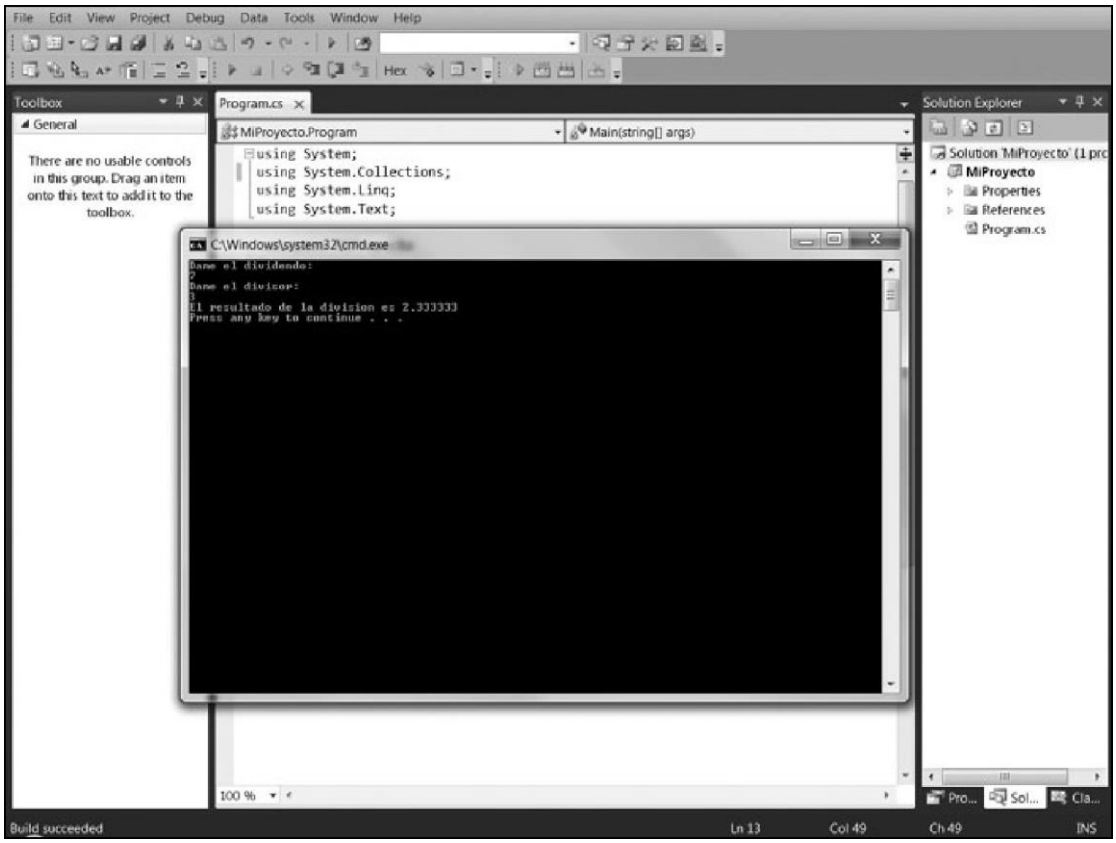

Figura 6. En este caso visto aquí, es ejecutada la división y a continuación se muestra el resultado.

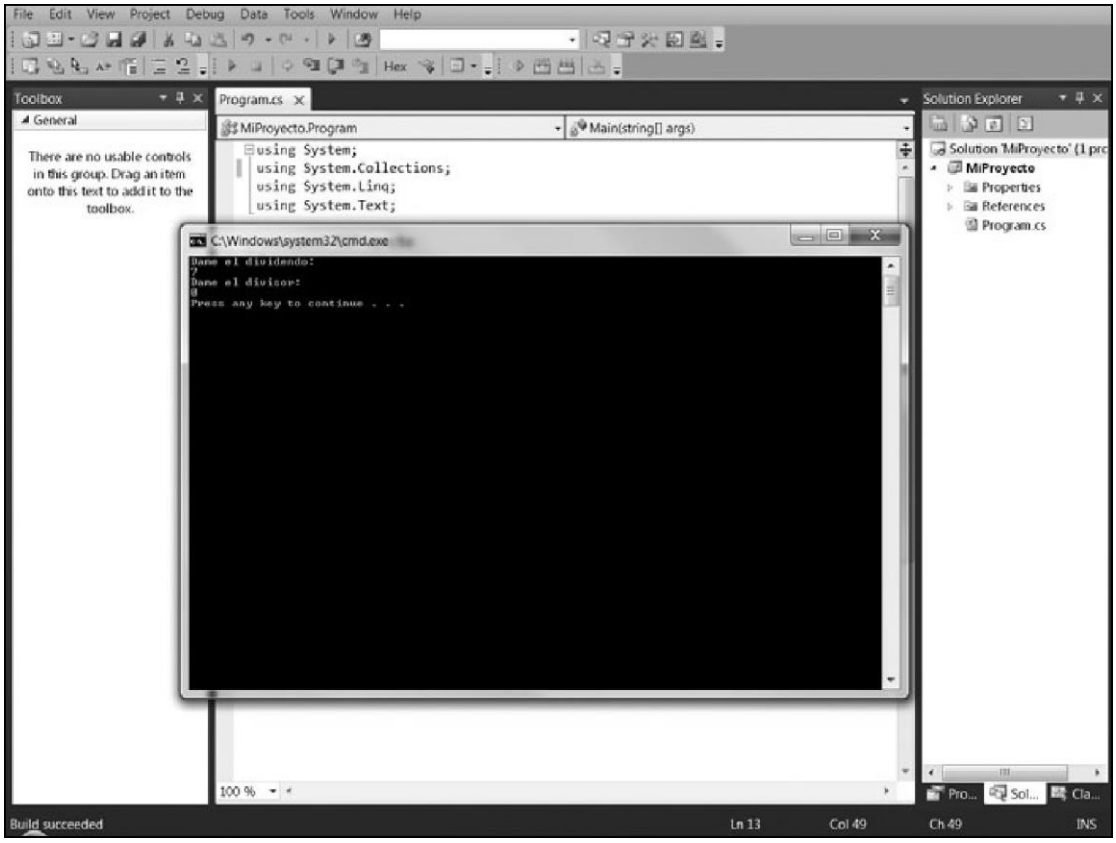

Figura 7. Como hemos colocado el valor de cero en el divisor, la división no se lleva a cabo.

USERS

El uso de eise

Con los programas anteriores con los que estuvimos trabajando, donde utilizamos la sentencia if, hemos visto que podemos indicar que determinada sentencia se ejecute cuando deseamos que una condición se cumpla. Sin embargo, puede ocurrir que a veces necesitamos que una sentencia se ejecute cuando se cumple la condición y que otra sentencia se ejecute cuando esa condición no se cumpla. Si nos encontramos en este caso, una forma de resolverlo sería colocar un if con una condición y luego otro if con una condición que sea complemento de la primera. Esto ya lo hemos hecho anteriormente en el programa de los números positivos y negativos. Sin embargo, existe otra forma más sencilla que la utilizada en este programa para hacerlo. Esta forma puede sernos de mucha utilidad para simplificar la lógica y que no nos veamos obligados de tener que colocar tantos if dentro de una sentencia. Para lograr esto haremos uso de else. Siempre utilizaremos else a continuación de una sentencia if, ya que else no se puede usar sólo. Tenemos que colocar nuestro código de la siguiente forma:

```
if(condición)
        Sentencial ;
else
        Sentencia2 ;
```

Si la condición es evaluada como verdadera, entonces se ejecuta la sentencia l , en cambio, cuando la condición se evalúa como falsa, se ejecuta la sentencia 2.

A continuación analizaremos cómo hacer uso de este caso para simplificar nuestro código. Para esto elegimos para representar este ejemplo el programa de los números positivos y negativos.

Lo primero que tenemos para analizar es el gráfico del diagrama de flujo, Y' como utilizamos una estructura o sentencia de tipo if-else, una de las salidas del rombo corresponde al código que se ejecuta cuando la expresión es evaluada como verdadera (if) y la otra salida es la que utilizamos para la sentencia que se ejecuta cuando la expresión es evaluada como falsa (else).

Siempre es conveniente verificar la ejecución del programa y colocar valores que puedan provocar

problemas con el fin de detectar cualquier error de lógica. Podemos probar valores positivos, negativos, el cero, fraccionarios, etcétera. Si encontramos problemas con los valores, podemos hacer uso del if para validar los valores antes de usarlos.

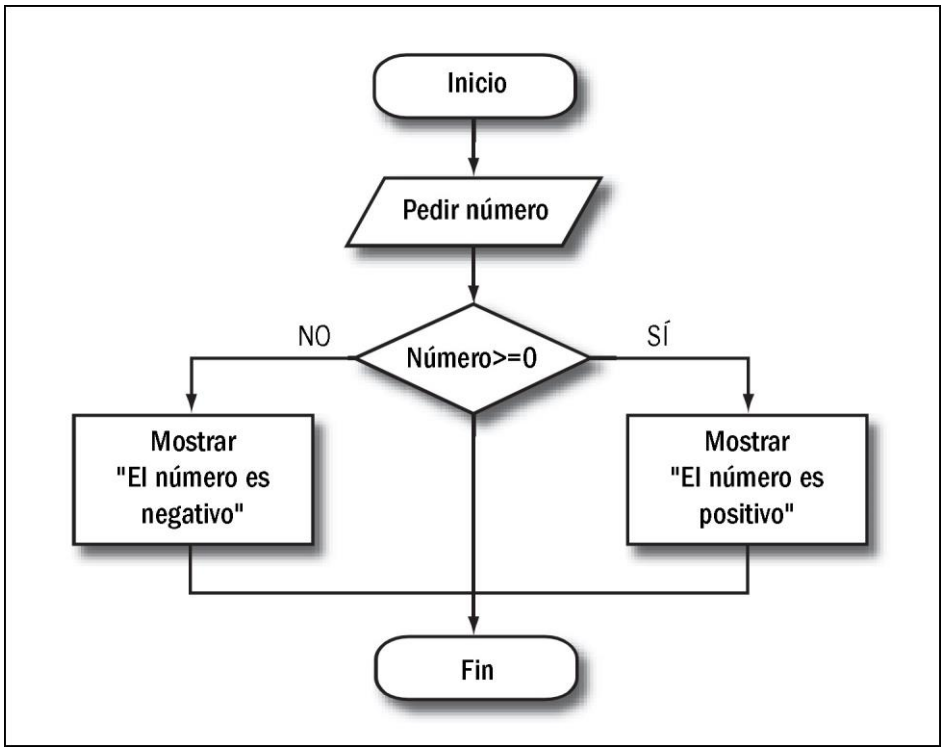

Figura 8. Aquí vemos claramente la ruta de if y la ruta de else.

El código de nuestro programa quedaría compuesto de la siguiente forma:

```
using System; using
System.C011ections.Generic;
using System.Text;

namespace AplicacionBase

class Program

    / / Esta es la función principal del programa
    / / Aquí inicia la aplicación static void
    Main(string[] args)
```

Cuando tenemos una estructura del tipo if-else, solamente la sentencia if o la sentencia else se ejecutará. En ningún caso se ejecutarán las dos sentencias. Solamente una de ellas puede ejecutarse.

Es importante tener esto en cuenta para evitar errores de lógica y no olvidar que else siempre requiere de un if.

```
int numero -- 0; // Donde guardamos el número string valor = // Para
guardar la cadena dada por el usuario
// Pedimos el numero
Console.Write("Dame un número entero:" ) ; valor =
Console.ReadLine() ; numero = Convert.ToInt32(valor) ; //
Convertimos la cadena a entero
// Hacemos uso de if con la expresión para el caso de los positivos if
(numero >= 0)
Console.WriteLine("E1 número {0} es positivo",
numero) ; ejecuta si se cumple numero>=0 else
Console.WriteLine("E1 numero {0} es negativo", numero) ;
    ejecuta si NO se cumple numero>=0
```

Podemos ver que el programa es más sencillo, se ha simplificado la lógica y esto lo hace más fácil de leer y mantener. Esto no significa que siempre debamos usar ifelse. Hay que hacer uso de él cuando tenemos algo que deseamos ejecutar cumpliendo previamente la condición especificada, y algo que deseamos ejecutar cuando ésta no se cumpla. Abusar de if-else o usarlo sin sentido complicará la lógica del programa, pero usarlo bien nos ayuda.

El uso de else con bloque de código

Al igual que con if, nosotros podemos colocar un bloque de código en else. Esto nos permitiría que más de una sentencia se ejecute cuando no se cumple la condición del if. No debemos olvidar colocar el bloque de código empezando con una llave, {, y finalizándolo con la llave de cierre }. Adentro del bloque de código podemos colocar cualquier sentencia válida de C#.

Por ejemplo, podemos modificar el programa de la división para hacer uso de la sentencia else. En este caso, lo que haremos será que, cuando el divisor sea igual a cero enviaremos a la consola un mensaje de error y cuando no lo sea, llevaremos a cabo la división. Primero veamos cómo es el diagrama de flujo con estos cambios. Es fácil notar la ruta de else.

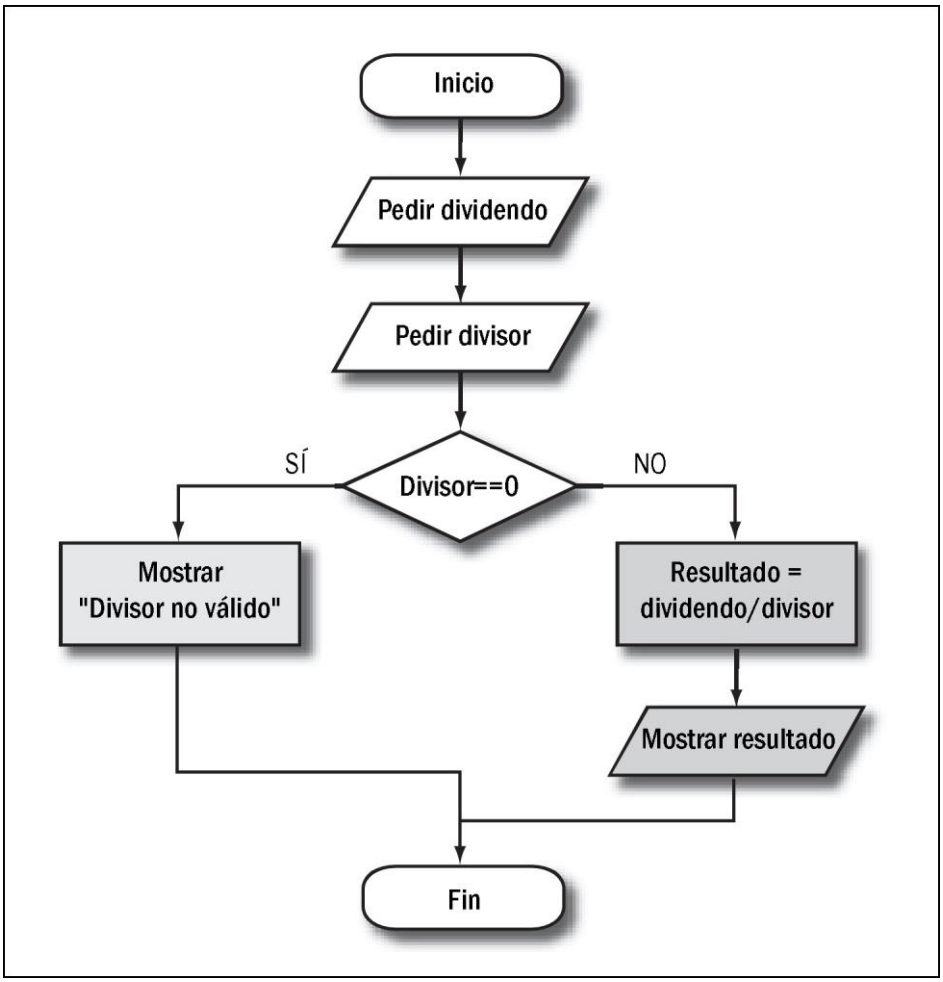

Figura 9. Aquí podemos ver el diagrama de flujo con las dos rutas de ejecución posibles, una para if y la otra para else.

El código del programa se modifica de la siguiente forma:

```
using System; using
System.C011ections.Generic;
using System.Text;

namespace AplicacionBase

class Program

    // Esta es la función principal del programa
    // Aquí inicia la aplicación static void
    Main(string[] args)

        // Variables necesarias float
        dividendo -- o.0f;
```

```
float divisor = 1 Of
float resultado --
o.0f; string valor =

// Pedimos el dividendo
Console.WriteLine("Dame el dividendo:" )
; valor = Console.ReadLine() ; dividendo =
Convert . ToInt32(va10r) ;

// Pedimos el divisor
Console.WriteLine("Dame el divisor: " ) ;
valor = Console.ReadLine() ; divisor =
Convert .ToInt32(va10r) ;

if (divisor 0)
        Console.WriteLine("El divisor no es válido") ;
else

        // Hacemos la operacion
        resultado = dividendo / divisor;

        // Mostramos el resultado
        Console.WriteLine("E1 resultado de la división es {0}" ,
            resultado) ;
```

De esta forma podemos manejar dos opciones, una para el divisor con el valor de cero y otra para cuando el divisor es válido.

Cuando hacemos uso de if-else, cualquiera de los dos o ambos pueden tener una sentencia o un bloque de código. No es necesario que ambos sean iguales. La selección entre la sentencia o el bloque de código nos la da el algoritmo y las necesidades de lógica que tengamos. Algunos programadores colocan el bloque de código por costumbre, aunque éste sólo tenga una sentencia.

Cómo usar if anidados

La sentencia o el bloque de código que ejecuta if puede ser cualquier sentencia válida o bloque de código válido en C#, esto incluye a otro if. Esto significa que nosotros podemos colocar if adentro del código a ejecutar de un if anterior. Cuando hacemos esto hay que ser cuidadosos para evitar errores de lógica. Esto se conoce como if anidados. Veamos un ejemplo de esto. Tenemos que hacer un programa que pedirá el tipo de operación aritmética que deseamos ejecutar y luego los dos números con los que llevar a cabo la operación. El resultado se desplegará en la pantalla. El programa tendrá el siguiente código:

```
using System; using
System.C011ections.Generic;
using System.Text;

namespace AplicacionBase

class Program
    {
     // Esta es la función principal del programa
     // Aqui inicia la aplicacion static void
     Main(string[] args)

          // Variables
          necesarias float a = 0
          Of float b = o.0f;
          float resultado = 0 Of
          string valor =    .int
          opcion = 0

          //                          Mostramos el menu
          Console. WriteLine("1-
                   WriteLine("2-      Console.Resta") ;
          Console.WriteLine("3- División") ;
          Console.WriteLine("4- Multiplicación") ;
          Console.Write("Que operación deseas
          hacer. valor=Cons01e. ReadLine() ; opcion
          Convert.ToInt32(valor);

          // Pedimos el primer número
          Console.Write("Dame el primer numero:" ) ;
```

```
valor = Console.ReadLine() ;
   - Convert.ToSing1e(va10r) ;
// Pedimos el segundo número
Console.Write("Dame el segundo numero:"
) ; valor = Console.ReadLine() ; b Convert .
ToSing1e(va10r) ;
    Verificamos para
suma if (opcion     1 )
resultado = a + b
    Verificamos para resta
if (opcion = 2) resultado
        b;
    Verificamos para division if (opcion = 3) if (b != 0)
// este if esta anidado resultado = a / b; else // Este
else pertenece al segundo if
                    Console.WriteLine("Divisor no valido") ;
    Verificamos para la multiplicacion if
(opcion       4) resultado -
    Mostramos el resultado
Console.WriteLine("El resultado es:         resultado) ;
```

El programa es sencillo, solamente necesitamos if para cada operación, pero para el caso de la división necesitaremos colocar otro if para verificar que el divisor no sea cero. Es decir que para la división tendremos if anidados. El diagrama de flujo de la aplicación se muestra en la siguiente figura.

Cuando colocamos un if adentro del código que puede ejecutar un if anterior, estamos diciendo que tenemos if anidados. Este tipo de estructura es muy útil y puede ayudarnos a optimizar los programas, pero debemos tener cuidado de usarla correctamente para evitar errores de lógica. Cuando se codifica hay que hacerlo en forma ordenada.

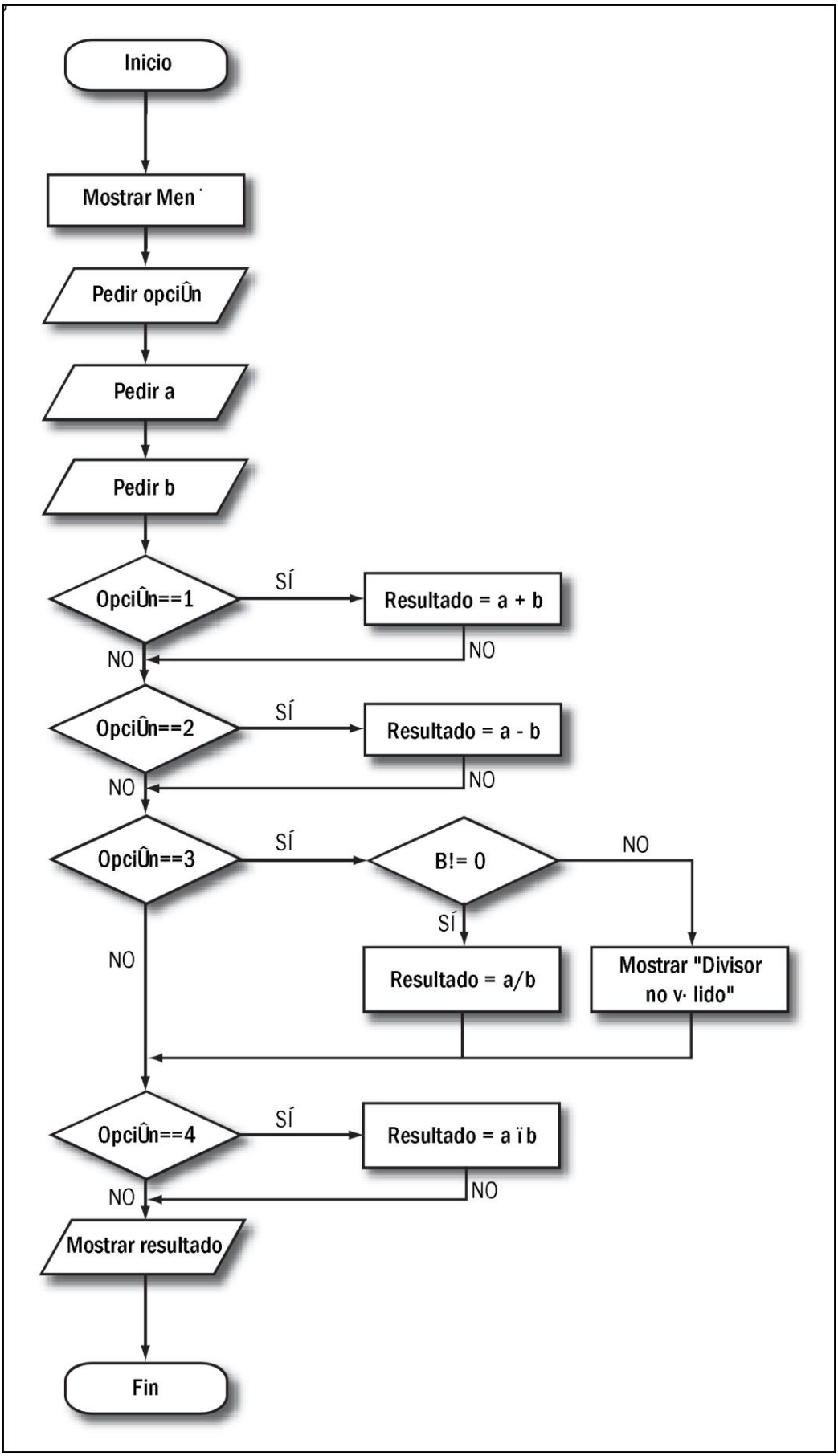

Figura 10. Éste es el diagrama de flujo. Es fácil observar en la división el anidamiento de if.

En el programa anterior observamos cómo se ha usado if anidado para la división, sólo tenemos que ser cuidadosos con else para poder saber a qué if pertenece. Si en nuestros programas hacemos uso de la indentación, es decir colocar espacios antes de la sentencia, podemos facilitar reconocer qué parte del código pertenece a qué sección. Algunos editores de código hacen esto por nosotros de forma automática y visualmente es más fácil reconocer el código.

Escalera de if-else

Otra estructura que se puede utilizar es la escalera de if-else. Una de sus funciones principales es optimizar la ejecución del código. Algunas veces son necesarias porque así lo requiere la lógica del programa. Al igual que con los if anidados, no hay que utilizarlos sin planeación, ya que al no programarlos correctamente, nos puede llevar a errores de lógica que pueden resultar difíciles de solucionar.

Ahora aprendamos cómo los podemos usar para que nuestro código se vuelva más eficiente. Si observamos el programa de las operaciones matemáticas, tenemos un if para cada operación. Esto significa que cuando nuestro programa se ejecuta, sin importar si se ha seleccionado la suma o la multiplicación, siempre se tienen que ejecutar al menos cuatro if. Los if pueden ser operaciones costosas ya que se tienen que evaluar expresiones.

La forma de optimizar esto es con una cadena de if-else. El concepto es sencillo y consiste en colocar un if en la sentencia del else.

Esto quedaría como muestra el siguiente código:

```
if(expresión 1 )
        Sentencia      1
else if(expresión 2)
                  Sentencia   2
        else if(expresión 3)
                    Sentencia
                    3
```

A veces sucede que cuando tenemos anidamiento, uno o varios de los if pueden tener else y luego no sabemos a quién pertenece. Else siempre pertenecerá al if más cercano que ya no tenga un else

asignado. La mejor forma de evitar problemas con esto es hacer uso de bloques de código, aunque solamente tengamos una sentencia en el bloque.

Como podemos ver se genera una figura que parece una escalera y por eso su nombree Modifiquemos el programa de las operaciones matemáticas. En primer lugar su diagrama de flujo queda de la forma que se muestra en la figura.

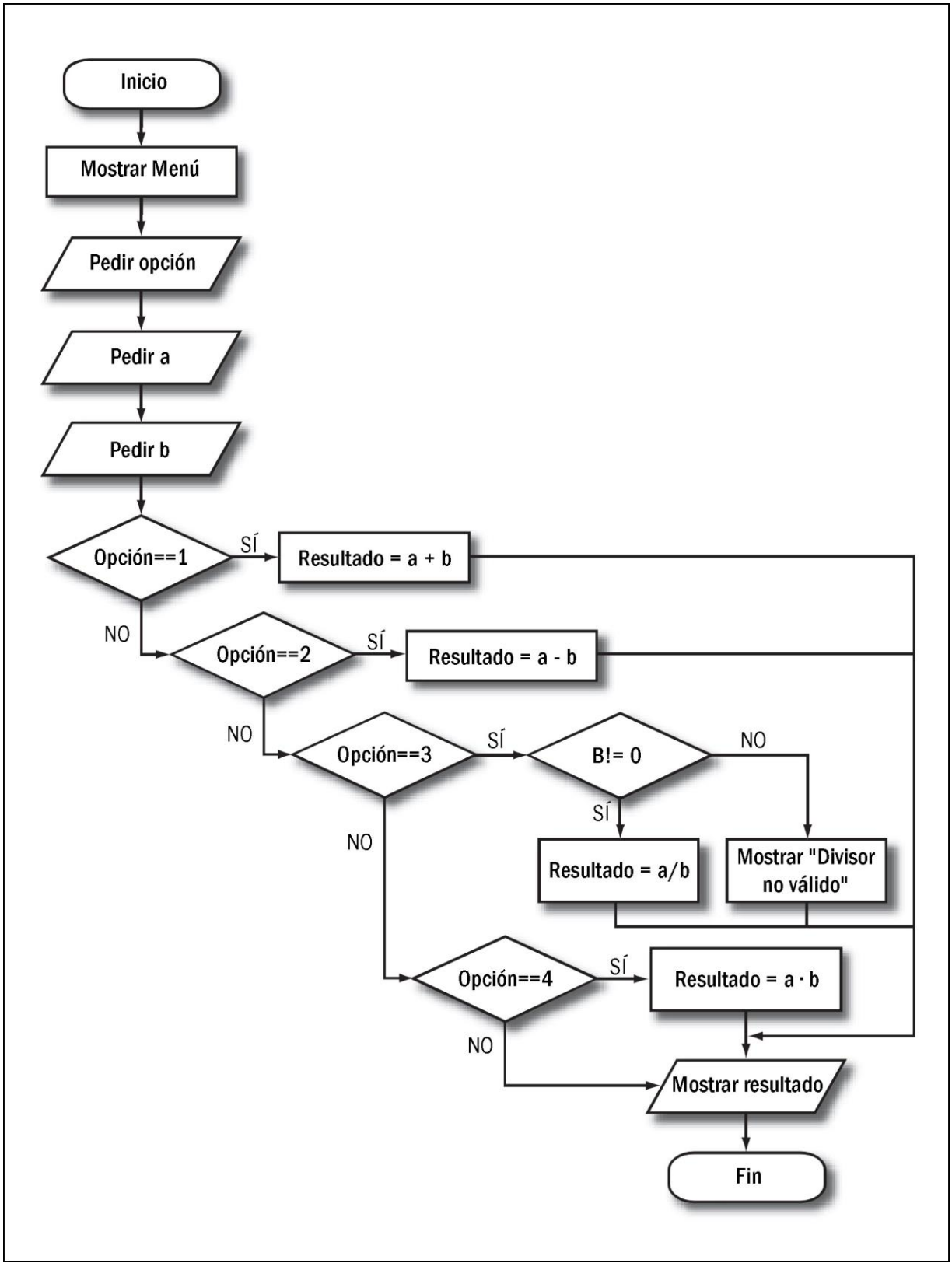

Figura 11. En este diagrama de flujo podemos observar nuestra escalera de if-else.

Modifiquemos el código del programa, colocándolo de la siguiente manera:

```
using System; using
System.C011ections.Generic;
using System.Text;

namespace AplicacionBase

    class Program

        // Esta es la función principal del programa
        // Aquí inicia la aplicación static
        void Main(string[] args)

            // Variables
            necesarias float a = 0
            Of float b = O Of
            float resultado --
            o.0f; string valor =
            int opcion -

            //                          Mostramos el menu
            Console. WriteLine("1-
                     WriteLine("2-      Console.Resta") ;
            Console.WriteLine("3- División") ;
            Console.  WriteLine("4-     Multiplicación")  ;
            Console.Write("Que operación deseas
            hacer. valor = Console. ReadLine() ; opcion =
            Convert . Tolnt32(va10r) ;

            // Pedimos el primer numero
            Console.Write("Dame el primer numero:" ) ;
            valor = Console.ReadLine() ;
              - Convert .ToSing1e(va10r) ;

            // Pedimos el segundo número
            Console.Write("Dame el segundo número:" ) ;
            valor = Console.ReadLine() ;
              Convert . ToSing1e(va10r) ;
```

```
/ / Verificamos para suma
if (opcion     1 ) resultado

/ / Verificamos para resta else if
(opcion 2) resultado = a ..

/ / Verificamos para division else if (opcion = 3) if (b != 0) /
        / este if esta anidado resultado _else / / Este else
        pertenece al segundo if
                                Console.WriteLine("Divisor  no
                                    válido") ;

                    / / Verificamos para la multiplicacion
                     else if (opcion 4) resultado - a

/ / Mostramos el resultado
Console.WriteLine("El resultado es: {0}" , resultado) ;
```

Si ejecutamos el programa, veremos que hace exactamente lo mismo que la versión anterior. ¿Pero dónde está la optimización? Si observamos el código, podemos encontrarla. Supongamos que el usuario ha seleccionado la opción l , es decir la suma.

Si la escalera if-else parece difícil de leer, podemos utilizar bloques de código para agrupar las secciones. Esto hará que entender y leer la lógica de la aplicación sea más fácil. La aplicación se puede resolver con if normales, no es forzoso utilizarlos. Se pueden utilizar cuando empezamos a tener más experiencia.

Con esto se cumple la condición del primer if y se ejecuta la operación de suma. Pero nosotros sabemos que se ejecuta if o else, nunca ambos. Si observamos, vemos que todas las demás operaciones están en la ruta de else, por lo que son ignoradas.

Antes teníamos que hacer cuatro if siempre, ahora solamente se hizo uno y los demás se evitaron, ahí es donde está la optimización. En otras palabras, en el programa anterior siempre eran cuatro if y en el nuevo, en el mejor de los casos tenemos un solo if, pero en el peor de los casos tenemos cuatro. Así es cómo se llevo a cabo la optimización.

Expresiones lógicas

Al igual que las expresiones aritméticas y relacionales, las expresiones lógicas tienen sus propios operadores. Estos son: y, o y no. En inglés los conocemos como: and, or y note

OPERADOR	SIGNIFICADO
&&	y
||	o
!	no

Tabla 2. Aquí tenemos los operadores lógicos utilizados en C#.

El uso de la conjunción

Empecemos a conocerlos. El primer operador es y, conocido como conjunción. Para usar este operador es necesario tener dos expresiones. Una expresión a la izquierda y la otra a la derecha. Las expresiones se evalúan devolviendo valores true o false. Con la conjunción, la expresión se evalúa como verdadera únicamente cuando ambas expresiones son verdaderas. La siguiente es la tabla de verdad para este operador:

P	Q	P&&Q
true	true	true
false	true	false
true	false	false
false	false	true

Tabla 3. La tabla de verdad de la conjunción.

Recordar la conjunción es fácil, únicamente es verdadera cuando ambas expresiones son verdaderas. Podemos recordar la regla o simplemente tener una impresión de la tabla a mano. Debemos recordar que se necesitan dos operandos, uno a la derecha y el otro a la izquierda. Con un solo operando no es posible hacerlo.

Supongamos que tenemos que decidir si debemos llenar el tanque de gasolina del vehículo. Esto lo hacemos si el tanque tiene menos del 50% y si la distancia a recorrer es de más de 200 km. Como vemos, tenemos que usar la conjunción ya que tenemos dos condiciones y ambas se deben cumplir para que suceda la carga de la gasolina. Si colocamos esto como expresión quedaría:

```
if(tanque < 50      distancia > 200)
      Cargar gasolina
```

Supongamos que tanque es 25 y distancia es 350, al evaluar 25 < 50 y 350 > 200 tenemos lo siguiente.

```
if(true     true)
      Cargar gasolina
```

En nuestra tabla sabemos que cuando ambas expresiones son verdaderas, entonces se cumple la conjunción y la expresión lógica nos evalúa a true.

```
if(true)
      Cargar gasolina
```

Por lo que cargaremos gasolina.

Ahora supongamos que tanque tiene el valor de 90 Las primeras evaluaciones nos darían: distancia es nuevamente 350.

```
if(false     tue)
      Cargar gasolina
```

Es fácil recordar la tabla de verdad de la disyunción. Esta es verdadera cuando cualquiera de las expresiones es verdadera. Ya que si una es verdadera o la otra es verdadera, entonces la expresión total es verdadera. También es bueno que tengamos esta tabla a mano hasta que aprendamos la disyunción de memoria.

Al ver nuestra tabla, vemos que false y true se evalúan como false, por lo que no realizaremos la carga de gasolina.

El uso de la disyunción

Para la disyunción hacemos uso del operador o. Este también necesita dos expresiones, una en el lado derecho y otra en el lado izquierdo. Esta disyunción tiene la siguiente tabla de verdad:

P	Q	P||Q
true	true	true
false	true	true
true	false	true
false	false	false

Tabla 4. La tabla de verdad de la disyunción.

Veamos un ejemplo con la disyunción. Tenemos que tomar la sombrilla si llueve o si hay mucho sol. La expresión podría quedar de la siguiente manera:

```
if(lluvia   true l l muchosol  t
      Tomar sombrilla
```

Supongamos que hay lluvia pero no hay sol, tendríamos la expresión:

```
if(true l l false)
      Tomar sombrilla
```

En este caso, si vemos la tabla de verdad, podemos observar que el resultado de evaluar la expresión nos da true, por lo que sí debemos tomar la sombrilla.

Ahora veamos qué sucede si no hay lluvia y no hay sol. La expresión quedaría como:

La negación simplemente intercambia true por false y false por true. Resulta fácil recordarla. Se invierte el valor del operando del lado derecho. No debemos olvidar que este operador solamente necesita un operando y siempre se encuentra del lado derecho. No hacerlo así puede originar problemas con nuestro programa.

```
if(false l l false)
      Tomar sombrilla
```

En este caso, la expresión se evalúa como false y no se toma la sombrilla.

El uso de la negación

Nos falta conocer un operador más, el de la negación. Ese operador es muy sencillo y solamente necesita hacer uso de un operando del lado derecho. Con estas condiciones dadas, esta expresión será negada por el operador.

	Negación
true	false
false	true

Tabla 5. La tabla de verdad de la negación.

Esto quiere decir que si el operando del lado derecho es true, el operador regresa false. Y si el operando del lado derecho es false, el operador regresa true.

Veamos un ejemplo para poder comprender mejor esto.
Tenemos una habitación con una bombilla eléctrica y supongamos que tenemos que prender la luz del cuarto cuando no es de día. Entonces nuestra expresión queda de la siguiente forma:

```
if( !dia   true)
      Prender la luz
```

Veamos qué sucede cuando la expresión dia tiene como valor verdadero. La expresión dia true se evalúa como true y luego éste se niega, lo que al final nos da como resultado un valor false. Como if recibe el valor false, se ejecutará la orden o el método prender la luz.
Pero si dia es false, entonces la expresión dia true se evalúa como false, que es negado e if recibe true. Como if recibe true, se ejecuta prender la luz.
Ahora podemos analizar un ejemplo de cómo hacer uso de una expresión lógica. Esto podemos aplicarlo en nuestro programa de las operaciones matemáticas. Podemos observar que la división se debe hacer cuando el usuario ha seleccionado entre las opciones presentadas, la número 3, y el divisor tiene un valor diferente de cero. Este es un caso en el que podemos hacer uso de la conjunción. Si procedemos a modificar la aplicación para realizar esto, nuestro diagrama de flujo debería modificarse, quedando similar a la forma que vemos a continuación:

USERS

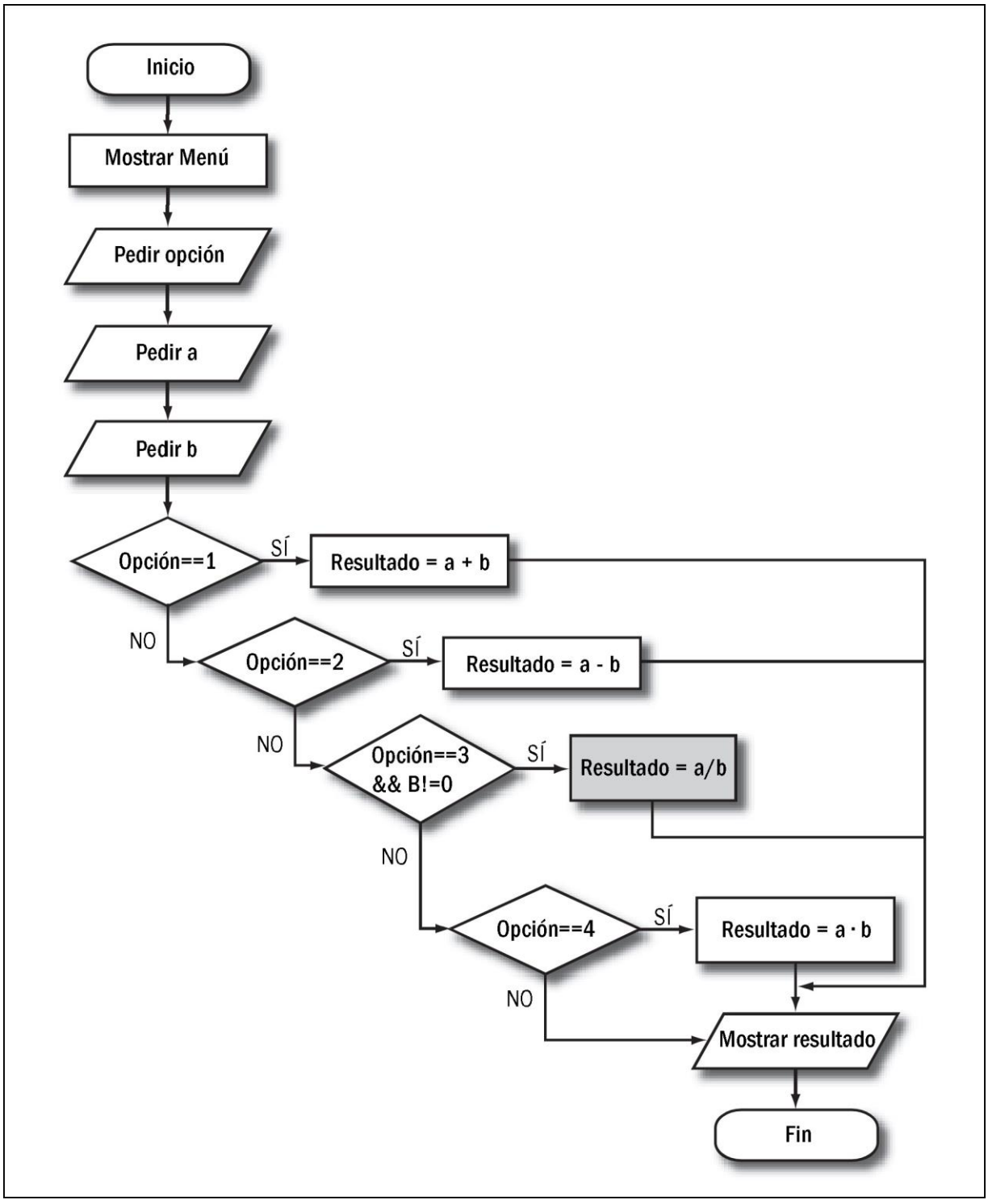

Figura 12. Podemos observar cómo la expresión lógica se encuentra dentro del rombo en la condición de la división.

El código del programa quedaría de la siguiente forma:

```
using System; using System.C011ections.Generic;
using System.Text;
```

```
namespace AplicacionBase

    class Program

        // Esta es la función principal del programa
        // Aquí inicia la aplicación static
        void Main(string[] args)

            // Variables
            necesarias float a =
            o.0f; float b = O Of
            float resultado --
            o.0f; string valor =
            int opcion -

            //                          Mostramos el menú
            Console.  riteLine("1-      Suma") ;
                      riteLine("2-      Console.Resta") ;
            Console.WriteLine("3- División") ;
            Console. W  r  iteLine("4-  Multiplicación") ;
            Console.Write("Que operación deseas hacer. ;
            valor = Console.ReadLine() ; opcion __
            Convert.ToInt32(valor);o

            // Pedimos el primer número
            Console.Write("Dame el primer numero:"
            ) ; valor = Console. ReadLine() ; a = Convert
            .ToSing1e(va10r) ;

            // Pedimos el segundo número
            Console.Write("Dame el segundo numero:"
            ) ; valor = Console.ReadLine() ; b = Convert
            .ToSing1e(va10r) ;

            // Verificamos para suma
            if (opcion  1 ) resultado

            // Verificamos para resta else
            if (opcion 2)
```

```
resultado =

/ / Verificamos para división else if (opcion  3       / /
Aquí usamos una expresión lógica resultado = a


/ / Verificamos para la multiplicación
else if (opcion = 4) resultado

/ / Mostramos el resultado
Console.WriteLine("El resultado es:        resultado) ;
```

Veamos otro ejemplo. Supongamos que tenemos que hacer un programa que nos indique si una persona puede conducir un automóvil, y las condiciones para que lo conduzca son que tenga más de 15 ó 18 años y que tenga permiso de sus padres. Si observamos el problema vemos que necesitamos dos variables, una para la edad y otra para el permiso de los padres. La expresión lógica podría quedar de la siguiente forma:

```
if(edad > 18 l l (edad > 15 && permiso                t
      Puede conducir
```

Vemos que en este caso usamos una expresión más compleja y al igual que con las expresiones aritméticas, hemos utilizado los paréntesis para ayudarnos y hacer que

Cuando tenemos expresiones complejas, para evitar errores es útil hacer una tabla de verdad de la expresión y probar todos sus casos. De esta forma podemos ver si funciona tal y como lo

esperábamos. La tabla de verdad debe ser similar a las que hemos utilizado y nos podemos apoyar en ella para encontrar el valor final de la expresión.

la lógica sea más fácil de leer. Nuestra expresión es una disyunción, en el lado izquierdo del or tenemos edad > 18, en el lado derecho tenemos una expresión

(edad > 15 && permiso true). La segunda expresión es una conjunción y deberá

evaluarse primero. El resultado que se obtenga será usado por la disyunción. Ahora podemos hacer el programa de ejemplo para esta expresión. Primero, observemos el diagrama de flujo a continuación:

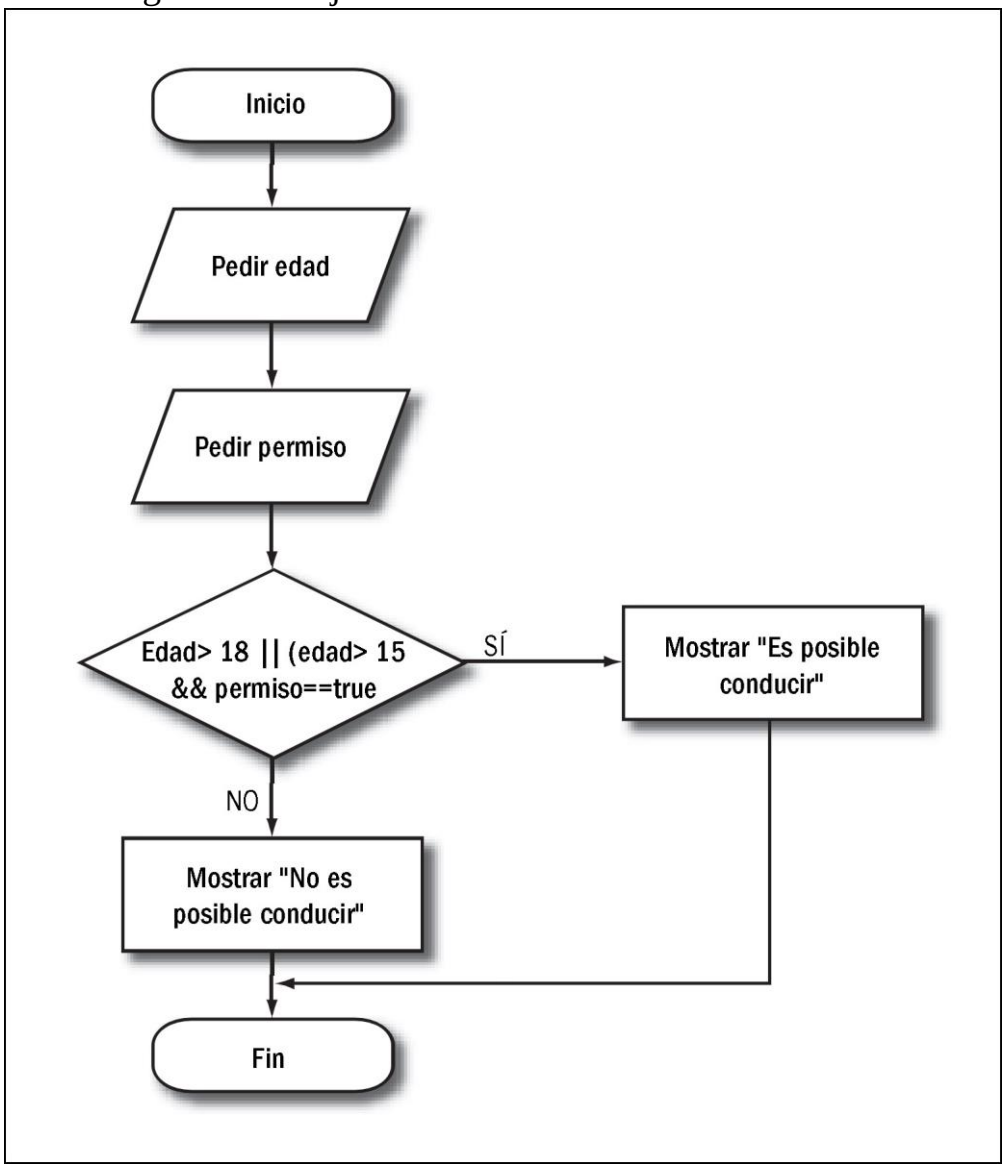

Figura 13. Al igual que en el caso anterior, debemos colocar la expresión dentro del rombo sin importar su complejidad.

El programa podría quedar de la siguiente manera:

```
using System; using
System.C011ections.Generic; using
System.Text;
```

USERS

```
namespace AplicacionBase

    class Program

        // Esta es la función principal del programa
        // Aquí inicia la aplicación static void
        Main(string[] args)

            // Variables necesarias
            int edad = o; bool
            permiso = false; string
            valor =

            // Obtenemos los datos
            Console.Write("Dame la edad: ▪
            valor = Console.ReadLine() ;

            edad -- Convert.T01nt32(va10r) ;

            Console.Write("Tiene permiso de los padres
 (true/false) .
            valor = Console.ReadLine() ;

            permiso = Convert .ToB001ean(va10r) ;

            // Verificamos que se cumpla la regla if(edad > 18 l l
            (edad > 15 && permiso      t
                Console.WriteLine("Es posible conducir") ;
            else o      n      s      ole.WriteLine(“No
            puedes conducir") ;
```

Con switch nosotros podemos comparar una variable contra diferentes valores. La variable puede ser

de tipo entero o cadena. El valor del caso debe ser apropiado para que funcione correctamente. De nuestro análisis conocemos los posibles valores para cada caso. El tipo de variable de comparación dependerá del algoritmo en particular.

```
            }
        }
    }
```

Ahora aprenderemos a usar otra estructura selectiva.

El uso de switch

En uno de los ejemplos anteriores, en los que realizamos las operaciones matemáticas, hemos observado que para cada operación debemos utilizar un if. Esto es correcto, y en varias ocasiones veremos que es necesario tomar diferentes opciones dependiendo del valor de una variable.

Como esto puede ocurrir frecuentemente, tenemos una estructura selectiva que nos ayuda y nos permite colocar el código para estos casos con facilidad. Esta estructura se conoce como switch. Para ésta necesitamos una variable y varios casos. Por ejemplo, en el programa de las operaciones, cada una de ellas es un caso. El valor de la variable se compara con un valor para cada caso. Si el valor coincide, entonces se empieza a ejecutar el código a partir de esa línea.

Cuando usamos switch es necesario colocar entre paréntesis la variable que utilizaremos para llevar a cabo las comparaciones. Luego tenemos que crear un bloque de código y colocar adentro de él los casos y el código a ejecutar para cada caso. Para indicar un caso, usamos case seguido del valor de comparación y dos puntos.

Existe un caso llamado default, que podemos utilizar si lo deseamos. Este caso siempre debe ser el último caso definido. Cuando la variable de comparación no ha encontrado su valor en ninguno de los casos, entonces se ejecuta el código del caso default. En los casos podemos colocar cualquier código C# que sea válido.

Para indicar dónde termina el código de un caso debemos hacer uso de break, que nos permitirá salir de la ejecución de una estructura selectiva o repetitiva. Esto lo veremos con más detalle en otro capítulo. Si no hacemos uso de break y el caso está vacío, entonces el programa continuará ejecutando el código del próximo caso y así sucesivamente hasta el final del bloque del código que pertenece al switch.

Para evitar errores en el switch debemos recordar que todos los casos deben terminar con break o return. Si no lo hacemos así, el compilador nos puede dar problemas. El uso de return lo aprenderemos en el capítulo de funciones. Es bueno tener bien definidos los casos para evitar problemas de lógica.

Switch puede ser representado en el diagrama de flujo por medio del rombo. En su interior colocamos la variable de comparación y de las esquinas o los lados podemos sacar las diferentes rutas de ejecución que puede tener. Como ejemplo modificaremos el programa de las operaciones aritméticas para que haga uso del switch. La variable de comparación será la variable opción, ya que el valor de ésta será comparado para cada operación.

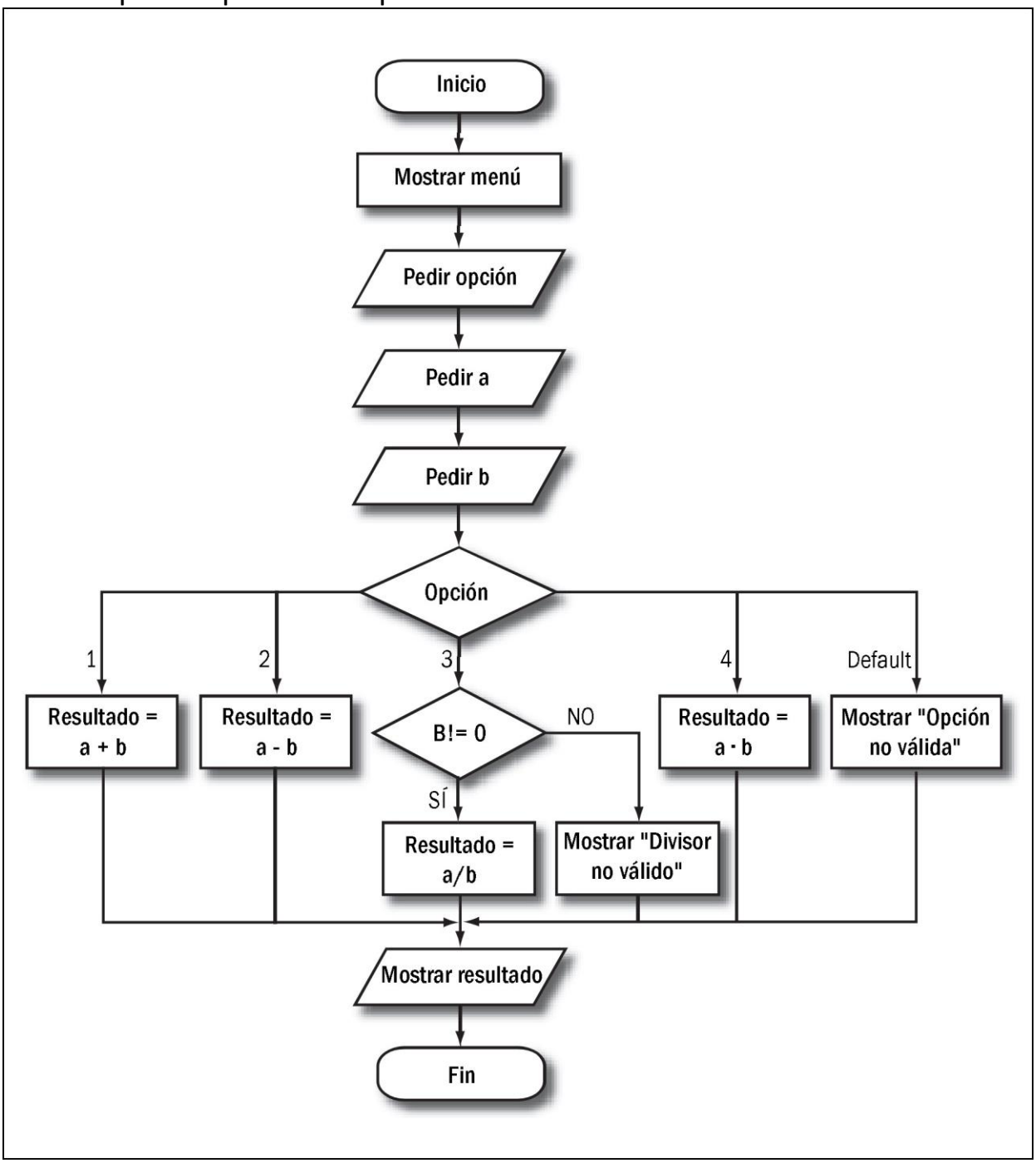

Figura 14. En este caso las diferentes rutas de ejecución del switch salen de los lados y las esquinas del rombo.

El código del programa quedará de la siguiente manera:

```
using System; using
System.C011ections.Generic;
using System.Text;

namespace AplicacionBase

    class Program

        // Esta es la función principal del programa
        // Aquí inicia la aplicación static void
        Main(string[] args)

            // Variables
            necesarias float a =
            o.0f; float b = O Of
            float resultado --
            o.0f; string valor =
            int opcion -

            //                          Mostramos el menú
            Console.  riteLine("1-      Suma") ;
                      riteLine("2-      Console.Resta") ;
            Console.WriteLine("3- División") ;
            Console.  r   iteLine("4-   Multiplicación")  ;
            Console.Write("Que operación deseas hacer. ,
            valor = Console. ReadLine() ; opcion = Convert .
            Tolnt32(va10r) ;

            // Pedimos el primer número
            Console.Write("Dame el primer numero:" ) ; valor
            = Console.ReadLine() ;
```

Es posible encontrar diferentes programas de computadora que nos permitan crear diagramas de flujo. Estos programas nos pueden ayudar a hacer nuestros programas más rápido. Un programa que

hace esto es Visio de Microsoft, pero también se pueden encontrar programas gratuitos en varios sitios web en Internet como www.download.com.

do =
e al

do") ;

```
// Mostramos el resultado
Console.WriteLine("El resultado es:                resultado) ;
```

OSERS

Con esto hemos visto las estructuras selectivas básicas en C# y cómo podemos usarlas.

RESUMEN

Las estructuras selectivas nos sirven para llevar a cabo la toma de decisiones. Tenemos varias de estas estructuras: if, if-else y switch. Podemos hacer uso de las expresiones relacionales y lógicas en if. Esto nos permite indicar de forma precisa la condición que se debe cumplir para que se ejecute determinado código. El uso de else nos permite tener código que se puede ejecutar cuando la condición no se cumple. Para comparar una variable contra diferentes valores y ejecutar un código en particular cuando su contenido es igual a un determinado valor hacemos uso del switch. Es importante no olvidar el uso del break al finalizar el código de cada caso.